Programming Assignment 1: NP-completeness

Revision: July 27, 2018

Introduction

Welcome to the first programming assignment of the NP-complete Problems course at edX! In this programming assignment, you will be practicing reducing hard real-world problems to SAT problem, which can in turn often be solved efficiently in practice using specialized programs called SAT-solvers.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Reduce real-world problems to instances of classical NP-complete problems.
2. Design and implement efficient algorithms to reduce the following computational problems to SAT:
   (a) assigning frequencies to the cells of a GSM network;
   (b) determine whether it is possible to leave no signs of a party in the apartment;
   (c) determine whether there is a way to allocate advertising budget given a set of constraints.

Passing Criteria: 2 out of 3

Passing this programming assignment requires passing at least 2 out of 3 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Assign Frequencies to the Cells of a GSM Network **2**

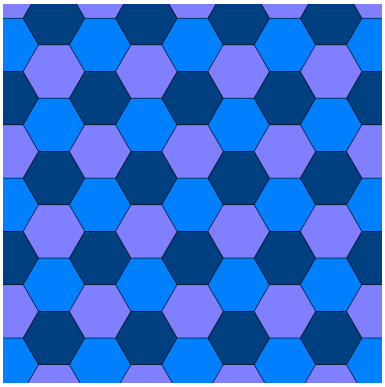
2 Cleaning the Apartment **5**

3 Advertisement Budget Allocation **8**

1 Assign Frequencies to the Cells of a GSM Network

Problem Introduction

In this problem, you will learn to reduce the real-world problem about assigning frequencies to the transmitting towers of the cells in a GSM network to a problem of proper coloring a graph into 3 colors. Then you will design and implement an algorithm to reduce this problem to an instance of SAT.



Problem Description

Task. GSM network is a type of infrastructure used for communication via mobile phones. It includes transmitting towers scattered around the area which operate in different frequencies. Typically there is one tower in the center of each hexagon called "cell" on the grid above — hence the name "cell phone". A cell phone looks for towers in the neighborhood and decides which one to use based on strength of signal and other properties. For a phone to distinguish among a few closest towers, the frequencies of the neighboring towers must be different. You are working on a plan of GSM network for mobile, and you have a restriction that you've only got 3 different frequencies from the government which you can use in your towers. You know which pairs of the towers are neighbors, and for all such pairs the towers in the pair must use different frequencies. You need to determine whether it is possible to assign frequencies to towers and satisfy these restrictions.

This is equivalent to a classical graph coloring problem: in other words, you are given a graph, and you need to color its vertices into 3 different colors, so that any two vertices connected by an edge need to be of different colors. Colors correspond to frequencies, vertices correspond to cells, and edges connect neighboring cells. Graph coloring is an NP-complete problem, so we don't currently know an efficient solution to it, and you need to reduce it to an instance of SAT problem which, although it is NP-complete, can often be solved efficiently in practice using special programs called SAT-solvers.

Input Format. The first line of the input contains integers n and m — the number of vertices and edges in the graph. The vertices are numbered from 1 through n. Each of the next m lines contains two integers u and v — the numbers of vertices connected by an edge. It is guaranteed that a vertex cannot be connected to itself by an edge.

Constraints. $2 \le n \le 500$; $1 \le m \le 1000$; $1 \le u, v \le n$; $u \ne v$.

Output Format. You need to output a boolean formula in the conjunctive normal form (CNF) in a specific format. If it is possible to color the vertices of the input graph in 3 colors such that any two vertices connected by an edge are of different colors, the formula must be satisfiable. Otherwise, the formula must be unsatisfiable. The number of variables in the formula must be at least 1 and at most 3000. The number of clauses must be at least 1 and at most 5000.

On the first line, output integers C and V — the number of clauses in the formula and the number of variables respectively. On each of the next C lines, output a description of a single clause. Each clause has a form $(x_4\ OR\ \overline{x_1}\ OR\ x_8)$. For a clause with k terms (in the example, $k = 3$ for x_4, x_1 and x_8), output first those k terms and then number 0 in the end (in the example, output "4 −1 8 0"). Output each term as integer number. Output variables $x_1, x_2, \ldots, x_V$ as numbers $1, 2, \ldots, V$ respectively. Output

negations of variables $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_V}$ as numbers $-1, -2, \ldots, -V$ respectively. Each number other than the last one in each line must be a non-zero integer between $-V$ and V where V is the total number of variables specified in the first line of the output. Ensure that $1 \le C \le 5000$ and $1 \le V \le 3000$.

See the examples below for further clarification of the output format.

If there are many different formulas that satisfy the requirements above, you can output any one of them.

Note that your formula will be checked internally by the grader using a SAT-solver. Although SAT-solvers often solve instances of SAT of the given size very fast, it cannot be guaranteed. If you submit a formula which cannot be resolved by the SAT-solver we use under a reasonable time limit, the grader will timeout, and the problem won't pass. We guarantee that there are solutions of this problem that output formulas which are resolved almost instantly by the SAT-solver used. However, don't try to intentionally break the system by submitting very complex SAT instances, because the problem still won't pass, and you will violate edX Honor Code by doing that.

Time Limits.

language	C	C++	Java	Python	Haskell	JavaScript	Scala
time (sec)	1	1	1.5	5	1.5	2	5

Memory Limit. 512MB.

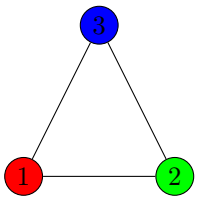
Sample 1.

Input:

```
3 3
1 2
2 3
1 3
```

Output:

```
1 1
1 -1 0
```



The input graph has just 3 vertices, so of course they all can be colored in different colors using only 3 colors. That's why we need to output a satisfiable formula. The formula in the output uses just 1 variable x_1 and 1 clause, and the only clause is $(x_1\ OR\ \overline{x_1})$ which is, of course, satisfiable: for any value of x_1, the boolean value of the formula is true. Note that you could output another satisfiable formula, like x_1 or $(x_1\ OR\ x_2\ OR\ x_3)\ AND\ (x_1\ OR\ x_2)$, or one of many others.

Sample 2.

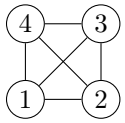
Input:

```
4 6
1 2
1 3
1 4
2 3
2 4
3 4
```

Output:

```
2 1
1 0
-1 0
```

Explanation:



The input graph has 4 vertices, and each pair of them is connected by an edge. In a proper coloring, all these vertices must be of different colors, but we have only 3 different colors, so it is impossible. Thus, we need to output an unsatisfiable formula. The formula in the output has 2 clauses with one variable, it is $(x_1)AND(\overline{x_1})$. Note that you could output another formula, like $(x_1\ OR\ x_2)\ AND\ (\overline{x_1})\ AND\ (\overline{x_2})$, or one of many other unsatisfiable formulas.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a procedure that outputs a fixed satisfiable formula. You need to change the main procedure to implement some reduction of the graph coloring problem to SAT problem if you're using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `gsm_network`

What To Do

Create a separate variable x_{ij} for each vertex i of the initial graph and each possible color j, $1 \le j \le 3$ which means "vertex i has color j". Think how to write down conditions like "each vertex has to be colored by some color" and "vertices connected by an edge must have different colors" with clauses of CNF using these variables.

Need Help?

Ask a question or check out the questions asked by other learners at this forum thread.

2 Cleaning the Apartment

Problem Introduction

In this problem, you will learn to determine whether it is possible to clean an apartment after a party without leaving any traces of the party. You will learn how to reduce it to the classic Hamiltonian Path problem, and then you will design and implement an efficient algorithm to reduce it to SAT.

Problem Description

Task. You've just had a huge party in your parents' house, and they are returning tomorrow. You need to not only clean the apartment, but leave no trace of the party. To do that, you need to clean all the rooms in some order. After finishing a thorough cleaning of some room, you cannot return to it anymore: you are afraid you'll ruin everything accidentally and will have to start over. So, you need to move from room to room, visit each room exactly once and clean it. You can only move from a room to the neighboring rooms. You want to determine whether this is possible at all.

This can be reduced to a classic Hamiltonian Path problem: given a graph, determine whether there is a route visiting each vertex exactly once. Rooms are vertices of the graph, and neighboring rooms are connected by edges. Hamiltonian Path problem is NP-complete, so we don't know an efficient algorithm to solve it. You need to reduce it to SAT, so that it can be solved efficiently by a SAT-solver.

Input Format. The first line contains two integers n and m — the number of rooms and the number of corridors connecting the rooms respectively. Each of the next m lines contains two integers u and v describing the corridor going from room u to room v. The corridors are two-way, that is, you can go both from u to v and from v to u. No two corridors have a common part, that is, every corridor only allows you to go from one room to one other room. Of course, no corridor connects a room to itself. Note that a corridor from u to v can be listed several times, and there can be listed both a corridor from u to v and a corridor from v to u.

Constraints. $1 \le n \le 30$; $0 \le m \le \frac{n(n-1)}{2}$; $1 \le u, v \le n$.

Output Format. You need to output a boolean formula in the CNF form in a specific format. If it is possible to go through all the rooms and visit each one exactly once to clean it, the formula must be satisfiable. Otherwise, the formula must be unsatisfiable. The sum of the numbers of variables used in each clause of the formula must not exceed 120 000.

On the first line, output integers C and V — the number of clauses in the formula and the number of variables respectively. On each of the next C lines, output a description of a single clause. Each clause has a form $(x_4\ OR\ \overline{x_1}\ OR\ x_8)$. For a clause with k terms (in the example, $k = 3$ for x_4, x_1 and x_8), output first those k terms and then number 0 in the end (in the example, output "4 -1 8 0"). Output each term as integer number. Output variables $x_1, x_2, \ldots, x_V$ as numbers $1, 2, \ldots, V$ respectively. Output negations of variables $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_V}$ as numbers $-1, -2, \ldots, -V$ respectively. Each number other than the last one in each line must be a non-zero integer between $-V$ and V where V is the total number of variables specified in the first line of the output. Ensure that the total number of non-zero integers in the C lines describing the clauses is at most 120 000.

See the examples below for further clarification of the output format.

If there are many different formulas that satisfy the requirements above, you can output any one of them.

Note that your formula will be checked internally by the grader using a SAT-solver. Although SAT-solvers often solve instances of SAT of the given size very fast, it cannot be guaranteed. If you submit a formula which cannot be resolved by the SAT-solver we use under a reasonable time limit, the grader will timeout, and the problem won't pass. We guarantee that there are solutions of this problem that output formulas which are resolved almost instantly by the SAT-solver used. However, don't try to intentionally break the system by submitting very complex SAT instances, because the problem still won't pass, and you will violate edX Honor Code by doing that.

Time Limits.

language	C	C++	Java	Python	Haskell	JavaScript	Scala
time (sec)	2	2	3	10	4	10	6

Sample 1.

Input:

```
5 4
1 2
2 3
3 5
4 5
```

Output:

```
1 1
1 -1 0
```

Explanation:

There is a Hamiltonian path $1-2-3-5-4$, so we need to output a satisfiable formula. The formula in the output uses just 1 variable x_1 and 1 clause, and the only clause is $(x_1\ OR\ \overline{x_1})$ which is, of course, satisfiable: for any value of x_1, the boolean value of the formula is true. Note that you could output another satisfiable formula, like x_1 or $(x_1\ OR\ x_2\ OR\ x_3)\ AND\ (x_1\ OR\ x_2)$, or one of many others.

Sample 2.

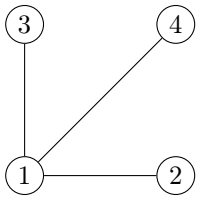
Input:

```
4 3
1 2
1 3
1 4
```

Output:

```
2 1
1 0
-1 0
```

Explanation:



There is no way to visit each room exactly once: either we don't visit one of the rooms 2, 3 or 4, or we visit room 1 at least twice. Thus, we need to output an unsatisfiable formula. The formula in the output has 2 clauses with one variable, it is $(x_1)AND(\overline{x_1})$. Note that you could output another formula, like $(x_1\ OR\ x_2)\ AND\ (\overline{x_1})\ AND\ (\overline{x_2})$, or one of many other unsatisfiable formulas.

Starter Files

The starter solutions for this problem read the data from the input, pass it to a procedure that outputs a fixed satisfiable formula. You need to change the main procedure to implement some reduction of the Hamiltonian path problem to SAT if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `cleaning_apartment`

What To Do

Create a separate variable x_{ij} for each vertex i and each position in the Hamiltonian path j. x_{ij} is true if vertex i is at the position j of Hamiltonian path.

Note that it is usually difficult to predict the running time of a SAT-solver on a CNF formula that you generate. In many cases, adding redundant clauses to a formula helps a SAT-solver to find a satisfying assignment (or to report that none exists) faster. For this reason, it is recommended that you include clauses describing all of the following constraints:

- Each vertex belongs to a path.
- Each vertex appears just once in a path. (Note that this restriction is already redundant: since we know that a path consists of n vertices and each vertex of a graph appears in it, it cannot be the case that some vertex appears more than once. Still, by adding such constraints you usually help a SAT-solver. Roughly, a solver does not need to spend time for figuring out that each vertex appears in a path just once; instead, it is given this information from the very beginning.)
- Each position in a path is occupied by some vertex.
- No two vertices occupy the same position of a path.
- Two successive vertices on a path must be connected by an edge. (In other words, if there is no edge $\{i, j\}$ in E, then for any k, it cannot be the case that both x_{ik} and $x_{i(k+1)}$ are True.)

Need Help?

Ask a question or check out the questions asked by other learners at this forum thread.

3 Advertisement Budget Allocation

Problem Introduction

In the previous programming assignment, you worked for an online advertisement system. In this programming assignment, you'll work for a big company that uses advertising to promote itself. You will need to determine whether it is possible to allocate advertising budget and satisfy all the constraints. You will learn how to reduce this problem to a particular type of Integer Linear Programming problem. Then you will design and implement an efficient algorithm to reduce this type of Integer Linear Programming to SAT.

Problem Description

Task. The marketing department of your big company has many subdepartments which control advertising on TV, radio, web search, contextual advertising, mobile advertising, etc. Each of them has prepared their advertising campaign plan, and of course you don't have enough budget to cover all of their proposals. You don't have enough time to go thoroughly through each subdepartment's proposals and cut them, because you need to set the budget for the next year tomorrow. You decide that you will either approve or decline each of the proposals as a whole.

There is a bunch of constraints you face. For example, your total advertising budget is limited. Also, you have some contracts with advertising agencies for some of the advertisement types that oblige you to spend at least some fixed budget on that kind of advertising, or you'll see huge penalties, so you'd better spend it. Also, there are different company policies that can be of the form that you spend at least 10% of your total advertising spend on mobile advertising to promote yourself in this new channel, or that you spend at least \$1M a month on TV advertisement, so that people always remember your brand. All of these constraints can be rewritten as an Integer Linear Programming: for each subdepartment i, denote by x_i boolean variable that corresponds to whether you will accept or decline the proposal of that subdepartment. Then each constraint can be written as a linear inequality. For example, $\sum_{i=1}^{n} spend_i \cdot x_i \le TotalBudget$ is the inequality to ensure your total budget is enough to accept all the selected proposals. And $\sum_{i=1}^{n} spend_i \cdot x_i \le 10 \cdot mobile$ corresponds to the fact that mobile advertisement budget is at least 10% of the total spending.

You will be given the final Integer Linear Programming problem in the input, and you will need to reduce it to SAT. **It is guaranteed that there will be at most 3 different variables with non-zero coefficients in each inequality of this Integer Linear Programming problem.**

Input Format. The first line contains two integers n and m — the number of inequalities and the number of variables. The next n lines contain the description of $n \times m$ matrix A with coefficients of inequalities (each of the n lines contains m integers, and at most 3 of them are non-zero), and the last line contains the description of the vector b (n integers) for the system of inequalities $Ax \le b$. You need to determine whether there exists a binary vector x satisfying all those inequalities.

Constraints. $1 \le n, m \le 500$; $-100 \le A_{ij} \le 100$; $-1\,000\,000 \le b_i \le 1\,000\,000$.

Output Format. You need to output a boolean formula in the CNF form in a specific format. If it is possible to accept some of the proposals and decline all the others while satisfying all the constraints, the formula must be satisfiable. Otherwise, the formula must be unsatisfiable. The number of variables in the formula must not exceed 3000, and the number of clauses must not exceed 5000.

On the first line, output integers C and V — the number of clauses in the formula and the number of variables respectively. On each of the next C lines, output a description of a single clause. Each clause has a form $(x_4\ OR\ \overline{x_1}\ OR\ x_8)$. For a clause with k terms (in the example, $k = 3$ for x_4, x_1 and x_8), output first those k terms and then number 0 in the end (in the example, output "4 −1 8 0"). Output each term as integer number. Output variables $x_1, x_2, \ldots, x_V$ as numbers $1, 2, \ldots, V$ respectively. Output negations of variables $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_V}$ as numbers $-1, -2, \ldots, -V$ respectively. Each number other than the last one in each line must be a non-zero integer between $-V$ and V where V is the total number of variables specified in the first line of the output. Ensure that $1 \le C \le 5000$ and $1 \le V \le 3000$.

See the examples below for further clarification of the output format.

If there are many different formulas that satisfy the requirements above, you can output any one of them.

Note that your formula will be checked internally by the grader using a SAT-solver. Although SAT-solvers often solve instances of SAT of the given size very fast, it cannot be guaranteed. If you submit a formula which cannot be resolved by the SAT-solver we use under a reasonable time limit, the grader will timeout, and the problem won't pass. We guarantee that there are solutions of this problem that output formulas which are resolved almost instantly by the SAT-solver used. However, don't try to intentionally break the system by submitting very complex SAT instances, because the problem still won't pass, and you will violate edX Honor Code by doing that.

Time Limits.

language	C	C++	Java	Python	Haskell	JavaScript	Scala
time (sec)	1	1	1.5	5	1.5	2	5

Memory Limit. 512MB.

Sample 1.

Input:

```
2 3
5 2 3
-1 -1 -1
6 -2
```

Output:

```
1 1
1 -1 0
```

Explanation:

Here we have two inequalities: $5 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \le 6$ and $(-1) \cdot x_1 + (-1) \cdot x_2 + (-1) \cdot x_3 \le -2$. The binary vector $x_1 = 0, x_2 = 1, x_3 = 1$ satisfies all the inequalities, so we need to output a satisfiable formula. The formula in the output uses just one variable x_1 and one clause, and the only clause is $(x_1\ OR\ \overline{x_1})$ which is, of course, satisfiable: for any value of x_1, the boolean value of the formula is true. Note that you could output another satisfiable formula, like x_1 or $(x_1\ OR\ x_2\ OR\ x_3)\ AND\ (x_1\ OR\ x_2)$, or one of many others. You **do not** have to make sure that the formula you output is satisfied only by the binary vectors which satisfy the given system of inequalities, but the intended solution ensures that.

Sample 2.

Input:

```
3 3
1 0 0
0 1 0
0 0 1
1 1 1
```

Output:

```
1 1
1 -1 0
```

Explanation:
There are three inequalities of the form $x_i \leq 1$. Of course, any binary vector satisfies all those inequalities, so we need to output a satisfiable formula. The formula in the output is the same as in the previous example.

Sample 3.

Input:

```
2 1
1
-1
0 -1
```

Output:

```
2 1
1 0
-1 0
```

Explanation:
There are two inequalities $x_1 \leq 0$ and $-x_1 \leq -1 \Rightarrow x_1 \geq 1$, but x_1 cannot be less than 0 and more than 1 simultaneously, so we need to output an unsatisfiable formula. The formula in the output has 2 clauses with one variable, it is $(x_1)AND(\overline{x_1})$. Note that you could output another formula, like $(x_1\ OR\ x_2)\ AND\ (\overline{x_1})\ AND\ (\overline{x_2})$, or one of many other unsatisfiable formulas.

Sample 4.

Input:

```
2 3
1 1 0
0 -1 -1
0 -2
```

Output:

```
2 1
1 0
-1 0
```

Explanation:
There are two inequalities: $x_1 + x_2 \leq 0$ and $(-1) \cdot x_2 + (-1) \cdot x_3 \leq -2 \Rightarrow x_2 + x_3 \geq 2$. From the first one, it follows for a binary vector x that $x_1 = x_2 = 0$. From the second one, it follows for a binary vector x that $x_2 = x_3 = 1$. But x_2 cannot be 0 and 1 simultaneously, so there is no binary vector x satisfying all the inequalities, and so we need to output an unsatisfiable formula. The formula in the output is the same as in the previous example.

Starter Files

The starter solutions for this problem read the data from the input, pass it to procedure that outputs a fixed satisfiable formula. You need to change the main procedure to implement some reduction of Integer Linear Programming problem to SAT if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `budget_allocation`

What To Do

In this case, you can use the binary vector x that you are looking for itself as the basis for the CNF formula. Think how to use the fact that there can be at most 3 non-zero coefficients in each inequality. Try thinking about assignments of all the corresponding variables that invalidate a particular inequality — how many different such assignments are there at most? You know that none of those assignments should hold if the binary vector x satisfies all the inequalities. Think how to write this condition compactly in the CNF form, then join those clauses for each inequality.

Need Help?

Ask a question or check out the questions asked by other learners at this forum thread.